**REGEX BACKREFERENCE**

What is a regex backreference? Backreferencing is all about **repeating** characters or substrings. There are occasions where you may want to isolate repeated substrings or characters within your search string. Backreferences work in conjunction with capturing groups to perform subsequent pattern matches. You will match the first occurrence of your regex with a capturing group, and then you can use backreferences to automatically locate subsequent matches. There are two types of backreferences, numbered backreferences and named backreferences.

I'll start off with a simple example, consider my grocery list: **"Apples, bananas, and tootsie rolls."**  
What are the characters that **repeat** in my grocery list? "A**pp**les, bananas, and t**oo**tsie ro**ll**s."

We can write a regex to find these repeating characters like this: **"(a)\\1"**, or **"(o)\\1"**, or **"(l)\\1"**. We can also simplify the regex by using the \\w metacharacter - "([\\w)\\1](file:///\\w)\\1)".

This is how a numbered backreference works: **(capturing group)**[**\\Number**](file:///\\Number)  
The number of the capturing group that you want to backreference (search repeated values) must match the automatically assigned capturing group number.  
regex **"(\\w)\\1"** will find pp, oo, and ll.

This is how a named backreference works: **(named group)\\k<name>**  
The named group that you want to backreference (search repeated values) will be prefixed with a **\\k** and then the group name inside chevrons **<name>**.  
regex **"(?<test>\\w)\\k<test>"** will find pp, oo, and ll.

I can best explain these concepts using source code examples.

|  |
| --- |
| **package** com.Soham;  **import** java.util.regex.\*;  **public class** Main {   **public static void** main(String[] args) {  Matcher m = **null**;   m = Pattern.*compile*(**"pp|oo|ll"**).matcher(**"Apples, bananas, and tootsie rolls."**);  **while** (m.find()) {  System.***out***.println(**"pp|oo|ll = "** + m.group());  }  System.***out***.println();   *// (p) - capturing group 1, (o) - capturing group 2, (l) - capturing group 3  //this sequence should always be same, as in the string or some wired output will arrive* m = Pattern.*compile*(**"(p)\\1|(o)\\2|(l)\\3"**).matcher(**"Apples, bananas, and tootsie rolls."**);  **while** (m.find()) {  System.***out***.println(**"(p)\\1|(o)\\2|(l)\\3 = "** + m.group());  }  System.***out***.println();   *//like ABOVE BUT, capturing group nos will be DIFFERENT* m = Pattern.*compile*(**"(p)\\1|(o)\\3|(l)\\2"**).matcher(**"Apples, bananas, and tootsie rolls."**);  System.***out***.println(**"capturing group nos will be DIFFERENT"**);  **while** (m.find()) {  System.***out***.println(**"(p)\\1|(o)\\2|(l)\\3 = "** + m.group());  }  System.***out***.println();   *//same as ABOVE* m = Pattern.*compile*(**"([a-z])\\1"**).matcher(**"Apples, bananas, and tootsie rolls."**);  **while** (m.find()) {  System.***out***.println(**"([a-z])\\1 = "** + m.group());  }  System.***out***.println();   *// \w - A word character, short for [a-zA-Z\_0-9]* m = Pattern.*compile*(**"(\\w)\\1"**).matcher(**"Apples, bananas, and tootsie rolls."**);  **while** (m.find()) {  System.***out***.println(**"(\\w)\\1 = "** + m.group());  }  System.***out***.println();   *// \k - everything that was matched so far from the overall match to be returned  // \w - A word character, short for [a-zA-Z\_0-9]  // <test> - is nothing but a named group* m = Pattern.*compile*(**"(?<test>\\w)\\k<test>"**).matcher(**"Apples, bananas, and tootsie rolls."**);  **while** (m.find()) {  System.***out***.println(**"(?<test>\\w)\\k<test> = "** + m.group());  }  System.***out***.println();  *// occurance of same characters in contagious position* m = Pattern.*compile*(**"([a-z][a-z])\\1"**).matcher(**"Apples, bananas, and tootsie rolls."**);  **while** (m.find()) {  System.***out***.println(**"([a-z][a-z])\\1 = "** + m.group());  }  System.***out***.println();   *// (t)(o)\2 - both 'O' s will be started with 't'  // (t)(o)\2\1 - "tootsie", the ending 'T' is nested, that is why \1 is used ONCE* m = Pattern.*compile*(**"(t)(o)\\2\\1"**).matcher(**"Apples, bananas, and tootsie rolls. SAAS"**);  **while** (m.find()) {  System.***out***.println(**"(t)(o)\\2\\1 = "** + m.group());  }  System.***out***.println();   *// INSTADE OF HARD CODING HERE, WE CAN USE '\W' (A word character, short for [a-zA-Z\_0-9])* m = Pattern.*compile*(**"(\\w)(\\w)\\2\\1"**).matcher(**"Apples, bananas, and tootsie rolls. SAAS"**);  **while** (m.find()) {  System.***out***.println(**"(\\w)(\\w)\\2\\1 = "** + m.group());  }  System.***out***.println();   *//BACKREFERANCE* m = Pattern.*compile*(**"(?<one>\\w)(?<two>\\w)\\k<two>\\k<one>"**).matcher(**"Apples, bananas, and tootsie rolls. SAAS"**);  **while** (m.find()) {  System.***out***.println(**"(?<one>\\w)(?<two>\\w)\\k<two>\\k<one> = "** + m.group());  }  System.***out***.println();   *// <[^>]{1,}> - First finds <,  // then [^>] (character class, THAT TAKES ANYTHING EXCEPT '>'  // {1,} - then it finds any no of characters --> QUANTIFIER,  // > - finally gets '>'* m = Pattern.*compile*(**"<[^>]{1,}>"**).matcher(**"<html> <body> Javac Java </body> </html>"**);  **while** (m.find()) {  System.***out***.println(**"<[^>]{1,}> = "** + m.group());  }  System.***out***.println();   *//SHORT HAND OF '<[^>]{1,}>' is '<[^>]+>'* m = Pattern.*compile*(**"<[^>]+>"**).matcher(**"<html> <body> Javac Java </body> </html>"**);  **while** (m.find()) {  System.***out***.println(**"<[^>]+> = "** + m.group());  }  System.***out***.println();    *// <[^>]+>.{0,}</[^>]+> stands for,  // <[^>]+> --> FINDS THE ARGUMENT LIKE ABOVE  // .(0,) --> . works as wild card, (0,) - that can be any length long  // </[^>]+> --> ENDS THE SAME* m = Pattern.*compile*(**"<[^>]+>.{0,}</[^>]+>"**).matcher(**"<html> <body> Javac Java </body> </html>"**);  **while** (m.find()) {  System.***out***.println(**"<[^>]+>.{0,}</[^>]+> = "** + m.group());  }  System.***out***.println();    *// SHORT HAND OF <[^>]+>.{0,}</[^>]+> IS <([^>]+)>.\*</\1>* m = Pattern.*compile*(**"<([^>]+)>.\*</\\1>"**).matcher(**"<html> <body> Javac Java </body> </html>"**);  **while** (m.find()) {  System.***out***.println(**"<([^>]+)>.\*</\\1> = "** + m.group());  }  System.***out***.println();   } } |
| **pp|oo|ll = pp**  **pp|oo|ll = oo**  **pp|oo|ll = ll**  **(p)\1|(o)\2|(l)\3 = pp**  **(p)\1|(o)\2|(l)\3 = oo**  **(p)\1|(o)\2|(l)\3 = ll**  **capturing group nos will be DIFFERENT**  **(p)\1|(o)\2|(l)\3 = pp**  **([a-z])\1 = pp**  **([a-z])\1 = oo**  **([a-z])\1 = ll**  **(\w)\1 = pp**  **(\w)\1 = oo**  **(\w)\1 = ll**  **(?<test>\w)\k<test> = pp**  **(?<test>\w)\k<test> = oo**  **(?<test>\w)\k<test> = ll**  **([a-z][a-z])\1 = anan**  **(t)(o)\2\1 = toot**  **(\w)(\w)\2\1 = toot**  **(\w)(\w)\2\1 = SAAS**  **(?<one>\w)(?<two>\w)\k<two>\k<one> = toot**  **(?<one>\w)(?<two>\w)\k<two>\k<one> = SAAS**  **<[^>]{1,}> = <html>**  **<[^>]{1,}> = <body>**  **<[^>]{1,}> = </body>**  **<[^>]{1,}> = </html>**  **<[^>]+> = <html>**  **<[^>]+> = <body>**  **<[^>]+> = </body>**  **<[^>]+> = </html>**  **<[^>]+>.{0,}</[^>]+> = <html> <body> Javac Java </body> </html>**  **<([^>]+)>.\*</\1> = <html> <body> Javac Java </body> </html>** |